**This manuscript is submitted to *IEE Proceedings – Computers and Digital Techniques***

**Paper Title:** Employing both inter-branch and intra-branch correlation to improve the accuracy of branch prediction

**Authors:** Meng-chou Chang, Ting-yu Chiu, and Chih-pei Chang

**Address:**
Chang Gung University
Department of Electrical Engineering
259 Wen-Hwa 1st Road, Kwei-Shan,
Tao-Yuan 333,
TAIWAN

**Corresponding Author:** Meng-chou Chang
**Email:** mchang@mail.cgu.edu.tw

# Employing both inter-branch and intra-branch correlation to improve the accuracy of branch prediction

Meng-chou Chang Ting-yu Chiu Chih-pei Chang

Department of Electrical Engineering

Chang Gung University, Tao-Yuan, Taiwan

Email: mchang@mail.cgu.edu.tw

## Abstract

Today's superscalar processors use branch prediction to reduce the influence of control hazards. Conventional two-level branch predictors make predictions based on either intra-branch correlation or inter-branch correlation. In the paper, the authors proposed two new branch predictors, called PGXg and PGAg, which make predictions by employing both intra-branch correlation and inter-branch correction. It is shown that the proposed branch predictors can achieve a better cost-performance ratio than conventional two-level branch predictors, such as PAg and gshare.

**Keyword:** superscalar processor, branch prediction, speculative execution, two-level branch predictor

## 1. Introduction

A superscalar processor has multiple functional units, allowing more than one instruction to be executed in parallel. However, the performance of a superscalar processor can not be improved unlimitedly by just providing enough functional units because the data and control dependences in a program add many constraints on the execution order of instructions. Most of today's superscalar processors use branch

prediction and speculative execution to alleviate the effects of conditional branches. When a superscalar processor fetches a branch instruction, it will predict the outcome of the branch and continue to execute the following instructions along the predicted path. The speculative results of pending instructions will be buffered in a dedicated hardware, such as the reorder buffer. When the outcome of the predicted branch become resolved, the speculative results will be committed if the prediction is proven to be correct; otherwise, the speculative results will be squashed and the processor will restart instruction fetching and execution from the misprediction point.

Branch predictions can be made in static or dynamic way. Static prediction schemes [1-4] predict the direction of a branch at compile time, while dynamic prediction schemes [5-18] predict the direction of a branch at run time. In general, dynamic prediction schemes are able to achieve higher prediction accuracy than static prediction schemes. Smith [1] proposed a dynamic branch prediction scheme that uses a table of 2-bit saturating up-down counters to predict the directions of branches. Each branch instruction is mapped via its address to a counter in the table. Whenever the result of a branch is resolved, its associated counter is updated according to the outcome of the branch. The counter is incremented by 1 if the outcome of the branch is taken, and is decremented by 1 if non-taken. When the processor fetches a branch, it predicts the result of the branch according to the value of the associated counter. If the most significant bit of the value is 1, the branch is predicted as taken; otherwise, the branch is predicted as non-taken.

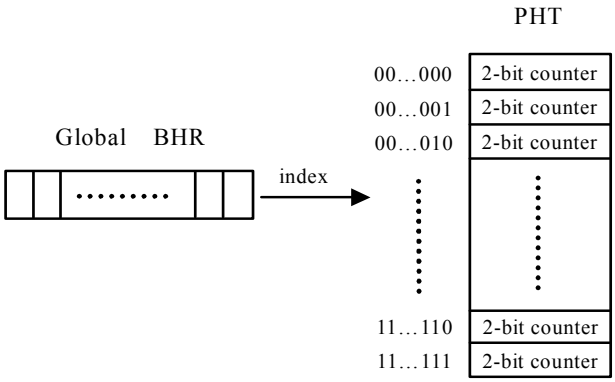



**Fig. 1** Structure of the branch predictor GAg.

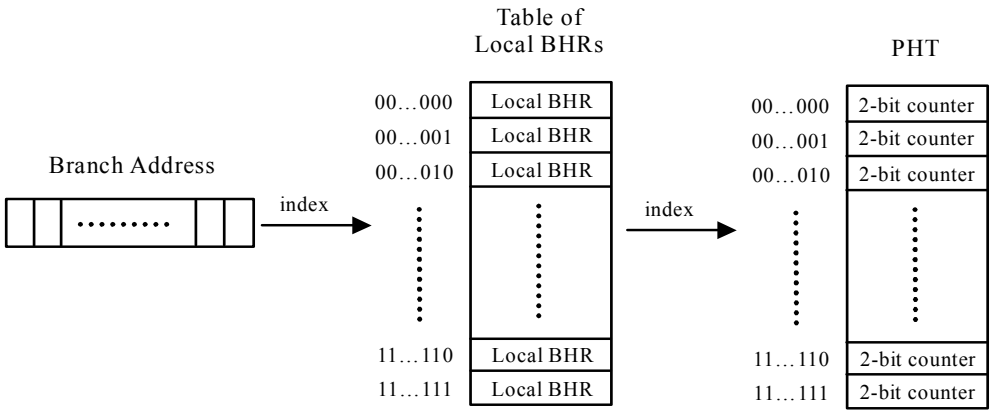



**Fig. 2** Structure of the branch predictor PAg.

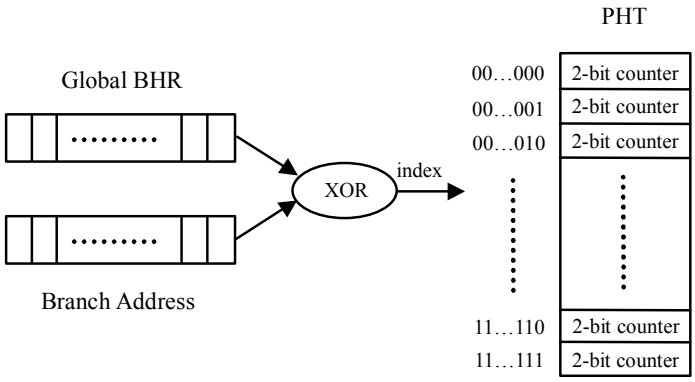



**Fig. 3** Structure of the branch predictor gshare.

Yeh and Patt [5] proposed the two-level branch prediction scheme. A two-level branch predictor is mainly composed of two components, BHR (branch history register) and PHT (pattern history table). BHR is used to record the outcomes of the most recently executed branches, and PHT is used to keep track of the most likely direction of a branch when a particular pattern is encountered in BHR. Two types of two-level branch predictors, GAg and PAg, are shown in Fig. 1 and Fig. 2. As shown in Fig. 1, the branch predictor GAg has only one BHR, the global BHR, which records the outcomes of all branches. The content of the global BHR represents the global branch history, which is used as an index to access the corresponding saturating counter in PHT. As shown in Fig. 2, the branch predictor PAg has a table of local BHRs, and the previous outcomes of a particular branch are recorded in its corresponding local BHR. The content of the local BHR represents the local branch history, which is used as an index to access the corresponding saturating counter in PHT. McFarling [6] proposed the branch predictor gshare, a variation of the global-history branch predictor. As shown in Fig. 3, the difference between gshare and GAg is that in gshare the global history is XORed with the branch address to form the index to PHT. It has been shown that gshare can achieve higher prediction accuracy because the addressing scheme of gshare can reduce the possibility of branch interferences in PHT.

In our previous research [17,18], we have proposed the branch predictor LGshare, which combines the intra-branch and inter-branch correlation to make predictions. In the paper, we proposed two new branch predictors, PGAg and PGXg, which also make predictions based on both intra-branch and inter-branch correlation and can achieve a better cost-performance ratio than conventional two-level branch predictors.

## 2. Importance of accurate branch prediction

Branch instructions and their target labels divide a program into basic blocks. A basic block is composed of a straight-line code sequence with no branches in except to the entry and no branches out except at the exit. Let $\beta$ denote the average dynamic branch frequency, then the average basic block size (i.e., the average number of instructions in a basic block) can be estimated as:

$$\begin{aligned} &Average\ basic\ block\ size \\ &= \frac{The\ number\ of\ total\ dynamic\ instructions}{The\ number\ of\ dynamic\ branch\ instructions} \\ &= \frac{The\ number\ of\ total\ dynamic\ instructions}{(The\ number\ of\ total\ dynamic\ instructions)\times \beta} \\ &= \frac{1}{\beta} \end{aligned} \qquad (1)$$

For typical MIPS programs the average dynamic branch frequency is often between *15%* and *25%* [19], so the average block size is between *4* and *6.67*. If a superscalar processor does not support branch prediction, it has to stall to wait for the outcome of the next branch every *4* to *6.67* instructions before it can continue to execute the following instructions in the next block. Since the size of a basic block is very small, the available instruction-level parallelism will be little, leading to underutilization of processor pipelines.

If a superscalar processor supports branch prediction and speculative execution, it can execute instructions across pending branches. Therefore, the processor can find more instructions to fill its pipelines, and a greater amount of instruction-level parallelism can be exploited. Let $\rho$ denote the accuracy of branch prediction, the average code size between two branch mispredictions, denoted by $\lambda$, can be estimated as:

$$
\begin{aligned}
&\textit{Average code size between two branch mispredictions } \lambda \\
&= \frac{\textit{The number of total dynamic instructions}}{\textit{The number of mispredicted branches}} \\
&= \frac{\textit{The number of total dynamic instructions}}{(\textit{The number of total dynamic instructions}) \times (\textit{branch frequency}) \times (\textit{misprediction rate})} \\
&= \frac{1}{\beta \times (1-\rho)}
\end{aligned}
\tag{2}
$$

Let branch frequency $\beta$ = 20%, then

$\lambda = 83$ if $\rho$ =94%

$\lambda = 100$ if $\rho$ =95%

$\lambda = 125$ if $\rho$ =96%

$\lambda = 167$ if $\rho$ =97%

$\lambda = 250$ if $\rho$ =98%

It is seen that the average code size between two branch mispredictions $\lambda$ increases rapidly with the increase of branch prediction accuracy $\rho$. For example, $\lambda$ has a improvement of 25% when $\rho$ is improved from 95% to 96%. Thus, the code segment in which a superscalar processor can exploit instruction-level parallelism without branch hazards is enlarged significantly even with a small improvement on the branch prediction accuracy. Moreover, the misprediction penalty is proportional to the misprediction rate, so improving the branch prediction accuracy can effectively reduce the misprediction penalty.

## 3. The branch predictors PGAg and PGXg

Conventional two-level branch predictors make predictions either based on inter-branch correlation or based on intra-branch correlation. However, the behavior of some branches in a program can not be well predicted by just employing one type of correlation.

The new branch predictors, PGAg and PGXg, use both inter-branch and

intra-branch correlation to make predictions. Fig. 4 shows the structure of the branch predictor PGAg. PGAg is composed of a global BHR, a table of local BHRs and a PHT. The global BHR records the global history of all branches, and the local history of a particular branch is recorded in its associated local BHR. The *g*-bit global branch history and the *l*-bit local branch history are concatenated to index the corresponding 2-bit saturating counter in PHT. In the way, the indexed counter records the branch tendency for a particular combination of the global history pattern and the local history pattern.

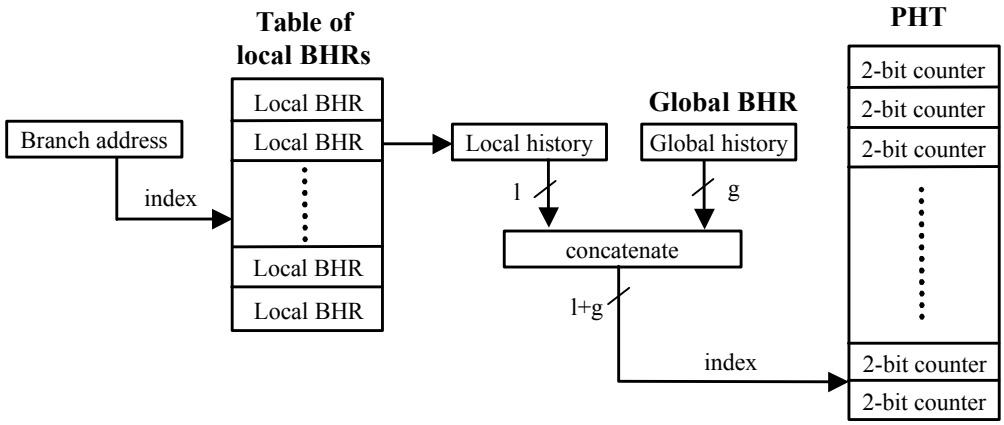



**Fig. 4** Structure of the branch predictor PGAg.

The structure of the branch predictor PGXg is shown in Fig. 5. The structure of PGXg is very similar to that of PGAg. The only difference is that the *g*-bit global branch history is first XORed with the branch address before it is concatenated with the *l*-bit local branch history. The operation of XOR might reduce the possibility of branch interferences in the pattern history table.

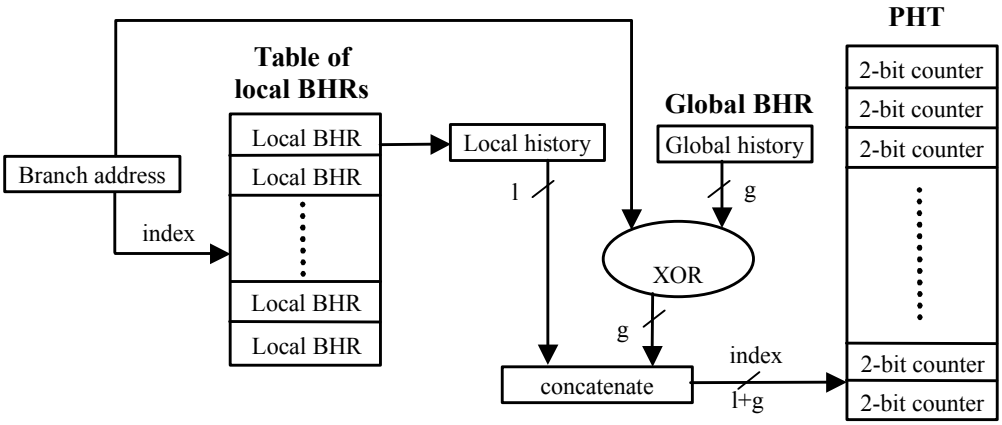



**Fig. 5** Structure of the branch predictor PGXg.

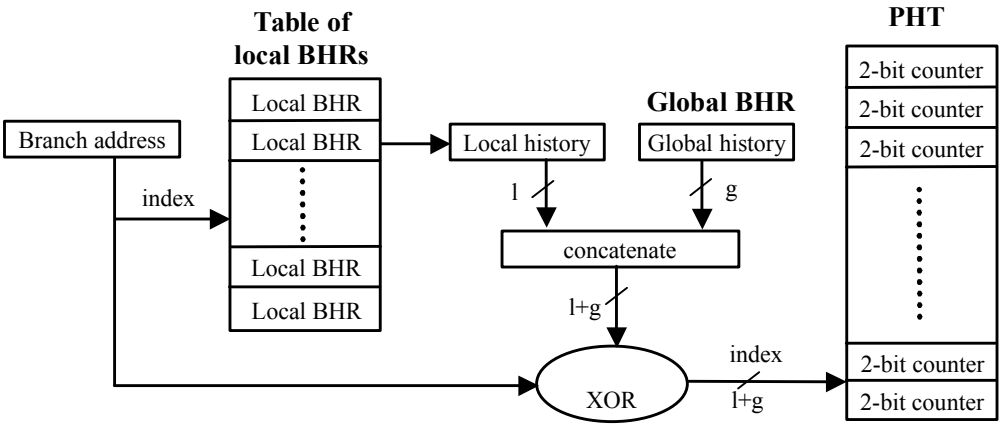



**Fig. 6** Structure of the branch predictor LGshare.

Both of the branch predictors PGAg and PGXg employ the combination of global and local branch history to access the PHT. Therefore, they predict the outcome of a branch based on both of the inter-branch and intra-branch correlation.

Fig. 6 shows the structure of another branch predictor, LGshare, which was introduced in our previous papers [17,18]. LGshare also makes predictions based on both inter-branch and intra-branch correlation. The performance of PGAg, PGXg, and LGshare will be compared in the next session.

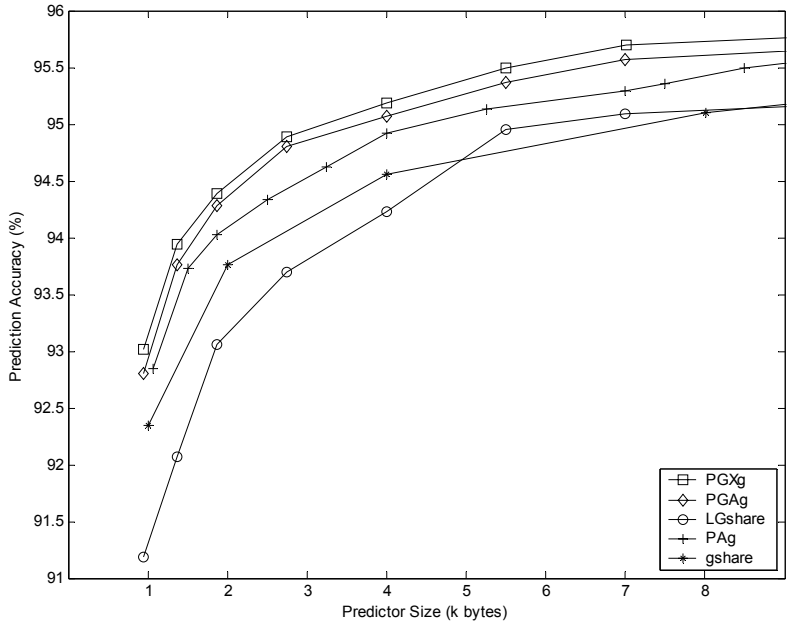
## 4. Simulation results

In this section, we will compare the performance of five two-level branch predictors, PAg, gshare, LGshare, PGAg, and PGXg. PAg makes predictions based on intra-branch correlation, and gshare make predictions based on inter-branch correlation. LGshare, PGAg, and PGXg make predictions by utilizing both inter-branch and intra-branch correlation.

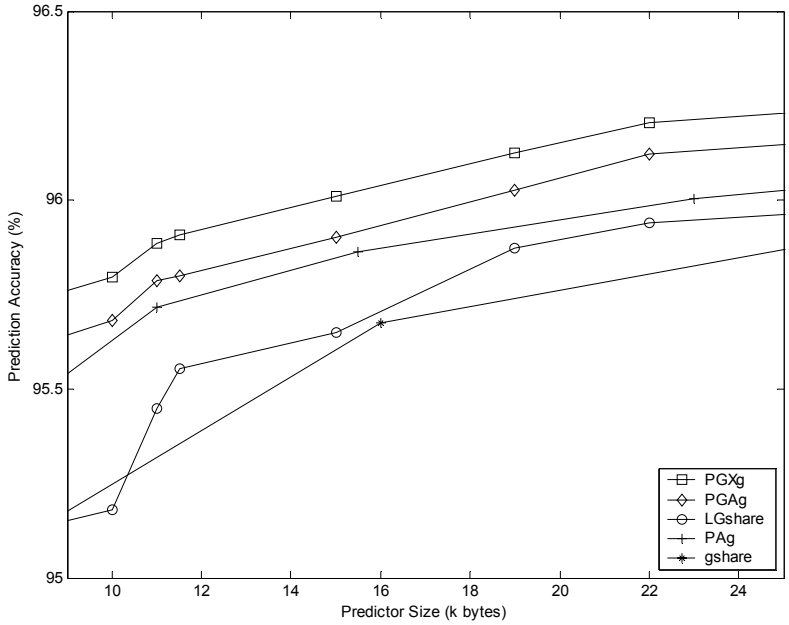
In the experiment, six benchmarks from the SPEC95int benchmark suit are used. The six benchmarks include *gcc, perl, m88ksim, vortex, li, and compress95*. All benchmarks are simulated until finish or for over 10000000 conditional branches.

Fig. 7 shows the performance comparisons of five branch predictors. The x-axis of the figure is the predictor size, which represents the hardware cost to implement the branch predictor. The formulas of predictor size are given in Table 1, where *g* denotes the length of the global BHR, *l* denotes the length of the local BHR, and *e* denotes the number of local BHRs in the predictor. Since the performance of LGshare, PGAg, and PGXg depend on their configuration (*l, g, e*), Table 2 gives the configurations of LGshare, PGAg, and PGXg corresponding to the points shown in Fig. 7.

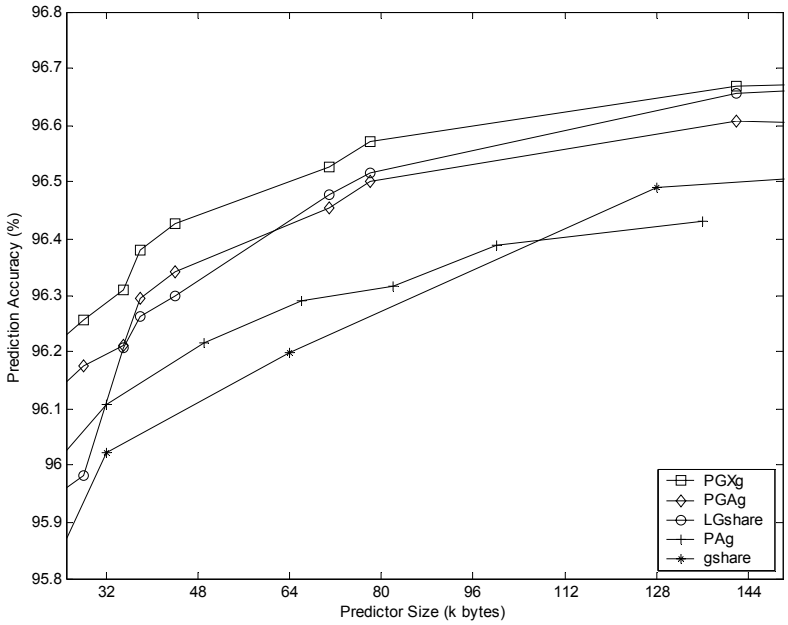
From Fig. 7, it is seen that PGXg has the best performance over other predictors. When the predictor size is small (smaller than 5 k-bytes), the relation of performance is PGXg > PGAg > PAg > gshare > LGshare. When the predictor size is medium (between 11 k-bytes to 30 k-bytes), the relation of performance is PGXg > PGAg > PAg > LGshare > gshare. When the predictor size is large (larger than 110 k-bytes), the relation of performance is PGXg > LGshare> PGAg > gshare > PAg.


Prediction Accuracy (%)
Predictor Size (k bytes)
PGXg
PGAg
LGshare
PAg
gshare

(a)


Prediction Accuracy (%)
Predictor Size (k bytes)
PGXg
PGAg
LGshare
PAg
gshare

(b)




(c)

**Fig. 7** Performance comparison of five branch predictors. (a) When the predictor size is between 0.8 k-bytes and 9 k-bytes. (b) When the predictor size is between 9 k-bytes and 25 k-bytes. (c) When the predictor size is between 25 k-bytes and 150 k-bytes.

**Table 1** The formulas of predictor size for various branch predictors.

| Predictor type | Predictor size (bits) |
|---|---|
| PAg | $g+2\times 2^{g}$ |
| gshare | $l\times e+2\times 2^{l}$ |
| LGshare | $l\times e+g+2\times 2^{(l+g)}$ |
| PGAg | |
| PGXg | |

**Table 2** The configurations of PGXg, PGAg, and LGshare corresponding to the points in Fig. 7.

| Configuration (*l, g, e*) | Predictor size (k bytes) | Prediction Accuracy (%) | | |
|---|---|---|---|---|
| | | PGXg | PGAg | LGshare |
| (7, 4, 512) | 0.937988 | 93.0163 | 92.8079 | 91.1901 |
| (7, 4, 1k) | 1.37549 | 93.9422 | 93.757 | 92.0682 |
| (7, 5, 1k) | 1.87561 | 94.3877 | 94.2785 | 93.0616 |
| (7, 5, 2k) | 2.75061 | 94.8872 | 94.8035 | 93.6963 |
| (8, 5, 2k) | 4.00061 | 95.1853 | 95.0734 | 94.2335 |
| (6, 8, 2k) | 5.50098 | 95.491 | 95.3635 | 94.9521 |
| (6, 8, 4k) | 7.0098 | 95.6936 | 95.5716 | 95.0936 |
| (6, 8, 8k) | 10.001 | 95.7973 | 95.6802 | 95.181 |
| (6, 9, 4k) | 11.0011 | 95.8842 | 95.7875 | 95.447 |
| (7, 8, 4k) | 11.501 | 95.909 | 95.7987 | 95.5535 |
| (7, 8, 8k) | 15.001 | 96.0097 | 95.9022 | 95.6482 |
| (6,10, 4k) | 19.0012 | 96.1235 | 96.0267 | 95.8731 |
| (6, 10, 8k) | 22.0012 | 96.204 | 96.1206 | 95.94 |
| (6, 10, 16k) | 28.0012 | 96.2558 | 96.1759 | 95.9814 |
| (6, 11, 4k) | 35.0013 | 96.3104 | 96.2118 | 96.2075 |
| (6, 11, 8k) | 38.0013 | 96.3807 | 96.2951 | 96.2637 |
| (6, 11, 16k) | 44.0013 | 96.4264 | 96.3414 | 96.2981 |
| (7, 11, 8k) | 71.0013 | 96.5261 | 96.4551 | 96.4786 |
| (7, 11, 16k) | 78.0013 | 96.572 | 96.5008 | 96.5152 |
| (7, 12, 16k) | 142.001 | 96.6699 | 96.6071 | 96.6565 |

When the predictor size is fixed at about 7 k-bytes, the misprediction rates of PGXg, PAg and gshare are 4.31%, 4.71%, and 4.89%, respectively. As branch misprediction penalty is proportional to branch misprediction rate, about 8.5% of branch misprediction penalty can be deleted when PGXg replaces the PAg branch prediction scheme, and about 11.9% of branch misprediction penalty can be deleted when PGXg replaces the gshare branch prediction scheme.

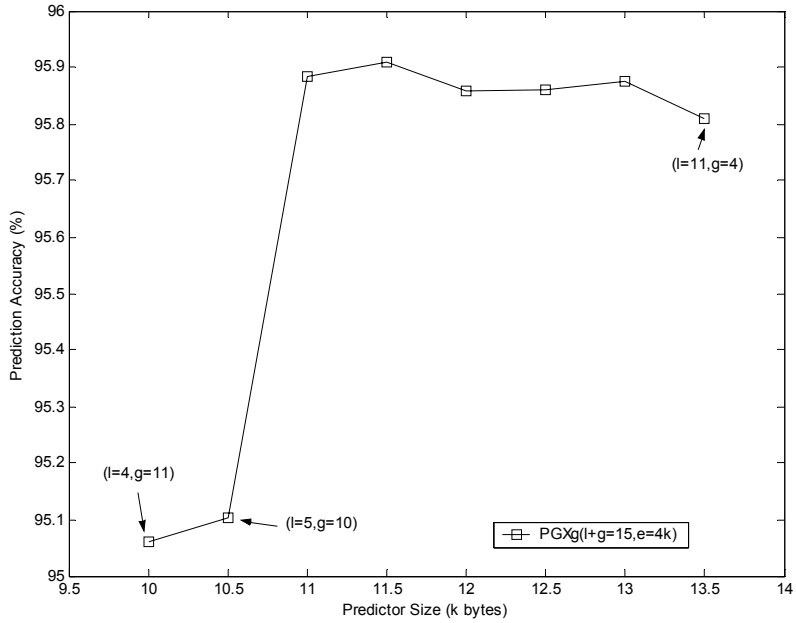
Since PGXg has better performance than other predictors, in the following we will concentrate only on PGXg. The performance of PGXg depends on both the length of

local branch history, *l*, and the length of global branch history, *g*. Fig. 8 shows how the performance of PGXg varies with *l* and *g* when the sum of *l* and *g* is fixed. In the case of Fig. 8 (a), it is seen that the best performance occurs at *l=7*. Although the predictor size of configuration PGXg(*l=7, g=8, e=4k*) is smaller than that of configuration PGXg(*l=11, g=4, e=4k*), configuration PGXg(*l=7, g=8, e=4k*) has better performance. For other cases with different combinations of (*l, g, e*), the performance curves are very similar to that of Fig. 8(a). That is, the performance increases as *l* grows, and it will reach a maximum, and then decreases slowly. The best value of *l* varies case by case. In our experiment (i.e., the predictor size is between 0.9 k-bytes and 150 k-bytes), the best value of *l* is between 6 and 8.

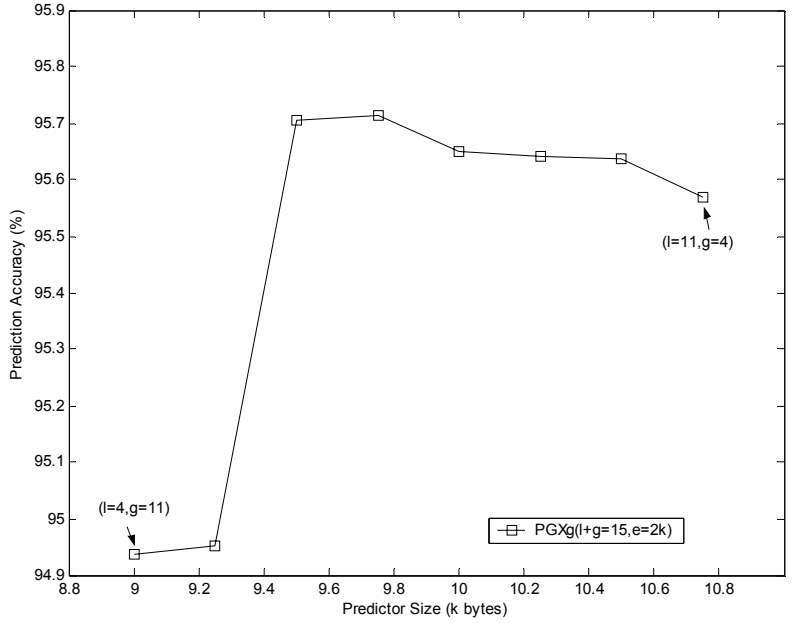
Fig. 9 shows how the performance of PGXg varies with *l, g,* and *e*. In Fig. 9(a), *l* is fixed at 7, *g* is varied from 1 to 12, and *e* is varied from 1k to 8k. When the hardware budge (predictor size) is increased, the number of local BHRs, *e*, should also be increased in order to achieve better performance. However, if the hardware budge is small, *e* should be small in order to reserve enough hardware budge for the PHT entries. In Fig. 9(b), *g* is fixed at 8, *l* is varied from 5 to 8, and *e* is varied from 1k to 8k. Similarly, in order to achieve better performance, *e* should be increased as the hardware budge grows.

The performance of PGXg depends on its configuration (*l, g, e*). From our experience, there is no simple and precise rule that can directly derive the best combination of (*l, g, e*) for a given hardware budge without any simulation. However, for a given hardware budge, many configurations of PGXg can achieve better performance than PAg and gshare if the hardware budge is not poorly allocated between PHT and the table of local BHRs. If a processor architect wants to incorporate the branch predictor PGXg in her/his design, s/he can run simulations with proper benchmarks to find the best combination (*l, g, e*) for a given hardware

budge, or s/he can find a proper configuration by looking up Table 2.

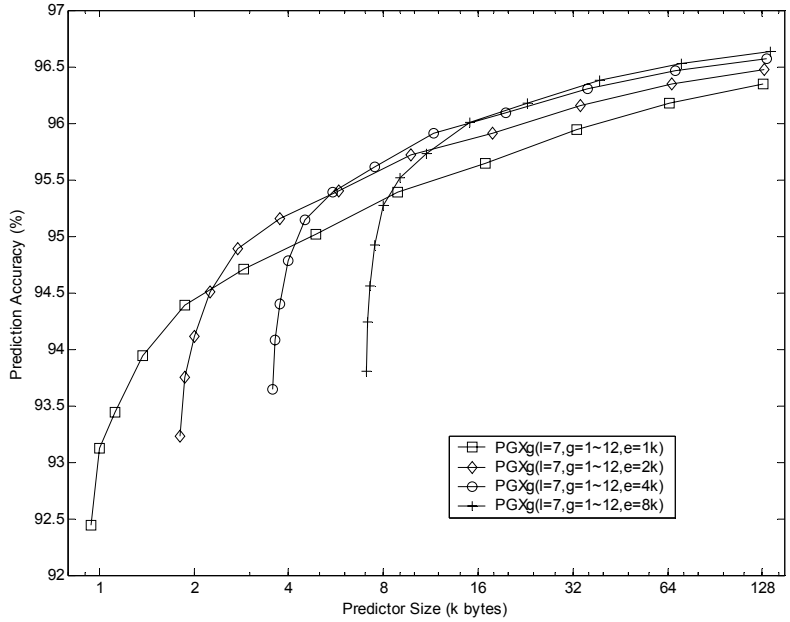



(a)

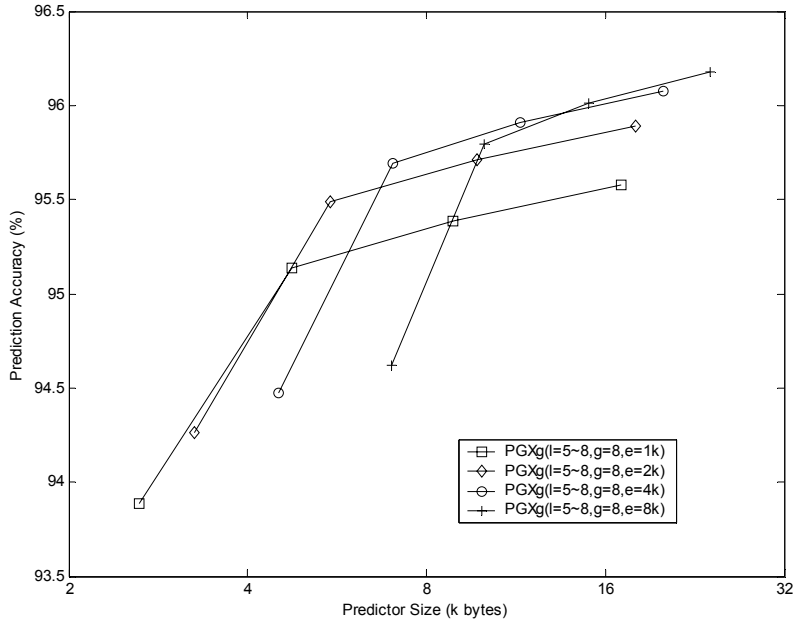



(b)

**Fig. 8** Effect of local history length *l* and global history length *g* on the performance of PGXg. (a) When *l*+*g*=*15* and *e*=*4k*. (b) When *l*+*g*=*15* and *e*=*2k*.




(a)




(b)

**Fig. 9** Variation of prediction accuracy with *l, g,* and *e*. (a) With *l* fixed, *g* and *e* varied. (b) With *g* fixed, *l* and *e* varied.

## 5. Conclusion

As the pipeline depth and issue rate of superscalar processors increase, the importance of an excellent branch predictor becomes more crucial to delivering the potential performance of a wise-issue, deep pipelined processor. We have proposed two branch predictors, PGAg and PGXg, which employ both the intra-branch correlation and the inter-branch correlation to improve the branch prediction accuracy. These two branch predictors can achieve better performance over conventional branch predictors such as PAg and gshare.

## Acknowledgement

This work is supported by the National Science Council, Taiwan, under Grant no. NSC92-2220-E-182-005.

## References


[1] Smith, J.E.: ‘A study of branch prediction strategies’. 8th Annual International Symposium on Computer Architecture, 1981, pp. 135-148

[2] McFarling, S., and Hennessy, J.: ‘Reducing the cost of branches’. 13th Annual International Symposium on Computer Architecture, 1986, pp. 396-403

[3] Fisher, J.A., and Freudenberfer, S.M.: ‘Predicting conditional branch directions from previous runs of a program’. 5th International Conference on Architectural Support for Programming Languages and Operating Systems, October 1992, Boston, pp. 85-95

[4] Deitrich, B.L., Chen, B.C., and Hwu, W.W.: ‘Improving static branch predictor in a compiler’. 1998 International Conference on Parallel Architectures and Compilation Techniques, 1998, pp. 214-221

[5] Yeh, T.-Y., and Patt, Y.N.: ‘Alternative implementations of two-level adaptive branch prediction’. 19th Annual International Symposium on Computer Architecture, 1992, pp. 124-134

[6] McFarling, S.: ‘Combining branch predictors’. Technical Report TN-36, Digital Western Research Laboratory, June 1993.

[7] Chang, P.-Y., Hao, E., Yeh, T.-Y., and Patt, Y.N.: ‘Branch classification: a new mechanism for improving branch predictor performance’. 27th Annual ACM/IEEE International Symposium on Microarchitecture, 1994, pp. 22-31

[8] Chang, P.-Y., Hao, E., and Patt, Y.N.: ‘Alternative implementations of hybrid branch predictors’. 28th ACM/IEEE Annual International Symposium on Microarchitecture, 1995, pp. 252-257

[9] Chang, P.-Y., Evers, M., and Patt, Y.N.: ‘Improving branch prediction accuracy by reducing pattern history table interference’. 1996 International Conference on Parallel Architecture and Compilation Techniques, 1996, pp. 48-57

[10] Lee, C.-C., Chen, I.K., and Mudge, T.N.: ‘The bi-mode branch predictor’. 30th Annual IEEE/ACM International Symposium Microarchitecture, 1997, pp. 4-13

[11] Sprangle, E., Chappell, R.S., Alsup, M., and Patt, Y.N.: ‘The agree predictor: a mechanism for

reducing negative branch history interference'. 24th Annual International Symposium on Computer Architecture, 1997, pp. 284-291

[12] Evers, M., Patel, S.J., Chappell R.S., and Patt, Y.N.: 'An analysis of correlation and predictability: what makes two-Level branch predictors work'. 25th Annual International Symposium on Computer Architecture, 1998, pp. 52-61

[13] Eden, A.N., and Mudge, T.: 'The YAGS branch prediction scheme'. 31st Annual IEEE/ACM International Symposium Microarchitecture, 1998, pp. 69-77

[14] Vintan, L.N., and Egan, C.: 'Extending correlation in branch prediction schemes'. 25th EUROMICRO Conference, 1999, pp. 441-448

[15] Manne, S., Klauser, A., and Grunwald, D.: 'Branch prediction using selective branch inversion'. 1999 International Conference on Parallel Architectures and Compilation Techniques, 1999, pp. 48-56

[16] Egan, C.: 'Dynamic Branch Prediction in High Performance Superscalar Processors.' PhD Dissertation, Department of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire, United Kingdom, August 2000

[17] Chang, M.-C., and Chou, Y.-W.: 'Improving the accuracy of branch prediction for superscalar processors by employing both the global and local branch history information.' 2nd International Conference on Parallel and Distributed Computing, Applications, and Techniques, 2001, pp. 252-258

[18] Chang, M.-C., and Chou, Y.-W.: 'Branch prediction using both global and local history information'. *IEE Proceedings-Computers and Digital Techniques*, Vol. 149, No. 2, March 2002, pp. 33-38

[19] Hennessy, J.L., and Patterson, D.A.: Computer Architecture – A Quantitative Approach. 3rd Ed., Morgan Kaufmann Publishers, 2003.